Informe Trabajo Práctico 1 y 2
Diseño de Compiladores I

Integrantes: Grupo 16

- Alberghini, Iván José Santiago ivan1alberghini@gmail.com
- Correa, Juan Pablo juampicorrea21@gmail.com
- Lugo, Brian Nicolás brai.lugo@gmail.com

Temas asignados:

1. **Enteros**: Constantes enteras con valores entre -2^{15} y $2^{15} - 1$. Estas constantes llevarán el sufijo "**_i**". Se debe incorporar a la lista de palabras reservadas la palabra **INTEGER.**

5. **Flotantes:** Números reales con signo y parte exponencial. El exponente comienza con la letra **f** (minúscula) y llevará signo. La parte exponencial puede estar ausente. Ejemplos válidos: 1. . 6 -1.2 3.f–5 2.f+34 2.5f-1 15. 0. Considerar el rango **1.17549435f-38 < x < 3.40282347f+38** U **-3.40282347f+38 < x < -1.17549435f-38** U **0.0** Se debe incorporar a la lista de palabras reservadas la palabra **FLOAT**.

11. **Sentencias declarativas:**

- Modificar los encabezados de declaraciones de procedimientos con la siguiente estructura: **PROC** ID (<lista_de_parametros>) **NA**=m, **SHADOWING** = <true_false> { … } Donde m será una constante de tipo entero (temas 1-2-3-4) y <true_false> puede ser **TRUE** o **FALSE**

- Ejemplos de declaraciones de procedimiento válidas:

PROC f1(**INTEGER** x, **FLOAT** y, **FLOAT** z) **NA** = 2, **SHADOWING** = **TRUE** {…}

PROC f2(**DOUBLE** a, **DOUBLE** b) **NA** = 1, **SHADOWING** = **FALSE** {…}

// Incorporar **NA**, **SHADOWING, TRUE y FALSE** a la lista de palabras reservadas.

Sentencias ejecutables:

- Sin cambios

12. Incorporar a la lista de palabras reservadas las palabras **WHILE** y **LOOP**.

19. Comentarios multilínea: Comentarios que comiencen con "/% " y terminen con "%/" (estos comentarios pueden ocupar más de una línea).

20. Cadenas de 1 línea: Cadenas de caracteres que comiencen y terminen con " ' " (estas cadenas no pueden ocupar más de una línea).

Introducción

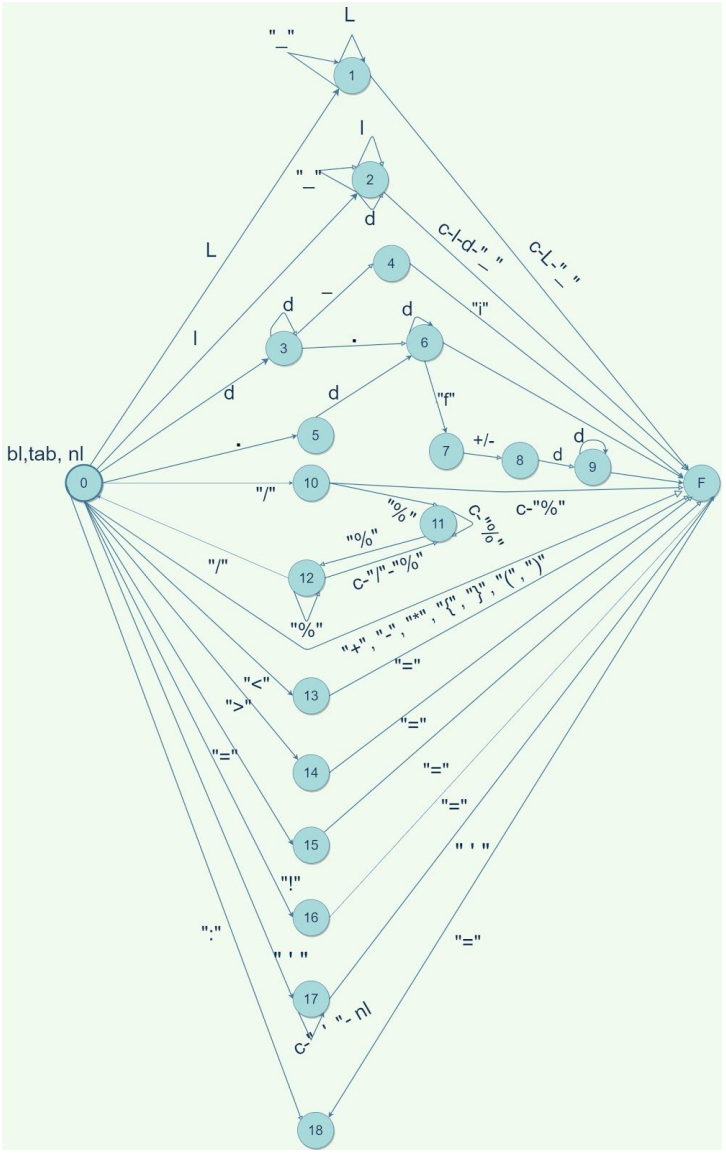
El objetivo de esta primera parte del trabajo es realizar un analizador léxico y a partir de este generar un sintáctico a partir de las especificaciones de una gramática acorde al lenguaje y los temas particulares asignados por la cátedra. Para la generación del analizador sintáctico utilizaremos la herramienta *YACC*.

Decisiones de diseño e implementación.

Para el desarrollo del trabajo práctico se utilizó el lenguaje de programación Java y por consiguiente la herramienta *YACC* para dicho lenguaje.
En cuanto a su implementación se tuvo en cuenta que el analizador léxico cuenta con dos partes fundamentales como lo son la matriz de transición de estados y la matriz de acciones semánticas, y la lógica necesaria para obtener los tokens uno a uno. También se consideró la tabla de símbolos que es una estructura común tanto al analizador léxico como al sintáctico.

Analizador Léxico

El objetivo de este analizador es leer los caracteres del código fuente y realizar el camino por el diagrama de transición de estados mientras realiza las acciones que se indican en la matriz de transición. Al llegar al estado final, el analizador devolverá el token que detectó y en caso de que este tenga un lexema, lo agrega a la Tabla de Símbolos, la cual implementamos con una HashTable para lograr que sea dinámica. También tenemos una variable que almacena la cantidad de referencias que tuvo un token, entonces, en caso de que un token ya exista en la tabla, procedemos a sumarle 1 a su cantidad de referencias. Una vez detectado un token, esperará la instrucción que le solicite detectar uno nuevo, así hasta el fin del programa. En caso de errores, los informará y continuará su ejecución con el fin de poder analizar todo el código en una única ejecución.
Como primer paso se realizó el diagrama de transición de estados, identificando previamente los tokens particulares del lenguaje. Luego se formaron los posibles caminos que estos podrían tener, los cuales los llevarían a un estado final, el cual indica que se detectó un token correctamente.



[Imagen 1] Diagrama de transición de estados

En base al diagrama, se creó la matriz de transición de estados. En esta matriz de transición, no todos los campos fueron completados. Estos, se tratarán mediante acciones semánticas para asegurar que el compilador no se detenga ante un estado inexistente.
Al tener los paréntesis (), como las llaves {}, el mismo comportamiento, se unificaron en un token.
La matriz de transición de estados fue implementada con una arreglo bidimensional que se podría visualizar como una matriz. Las filas de esta matriz corresponden a cada estado del diagrama de transición de estado, y las columnas a los diferentes caracteres que soporta dicho diagrama.
Cuando llega un caracter, se asoció su símbolo ASCII con un valor de la matriz (el valor con el que identificamos al caracter) y se utilizó ese valor asociado para calcular el próximo estado y las acciones semánticas correspondientes.

Luego se asoció a cada transición una acción semántica. Estas acciones semánticas fueron implementadas mediante una herencia, donde la clase madre es AccionSemantica y las clases que heredan de ella definen su funcionamiento en el método accionar.
Estas son:

- Inicializar Buffer: inicializa un buffer con el primer caracter leido
- Agregar Carácter: agrega el caracter leído al buffer
- Verificar KeyWord: verifica si el token detectado es una palabra clave
- Verificar LargoID: verifica el largo del token identificador
- Verificar Rango Integer: verifica el rango del token integer
- Verificar Rango Float: verifica el rango del token float
- Verificar Comparador: verifica si el token es un comparador
- Contar NL: añade uno al contador de saltos de línea
- Barra Divisora: Cuando se detecta que a pesar de haber leído una barra divisoria, el token no era un comentario, se devuelve como token la barra divisora
- Carácter Simple: Devuelve un caracter simple que de por sí es un token
- Cadena: Devuelve una cadena de caracteres
- Asignación: Verifica si el token es una asignación y sino se entrega el token asumiendo que el programador no incluyó el “=”

	l	L	d	_	%	+	-	"/"	*	{,},(,)	.	<	=	>	!	"	Bl/tab	nl	i	f	otro	$	:
0	IB	IB	IB			C	C	-	C	C	IB	IB	IB	IB	IB	IB	-	NL	IB	IB	C	C	IB
1	KW	AG	KW	AG	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW
2	AG	ID	AG	AG	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	AG	AG	ID	ID	ID
3			AG	-							AG												
4																			INT				
5			AG																				
6	F	F	AG	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	AG	F	F	F
7						AG	AG																
8			AG																				
9	F	F	AG	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
10	B	B	B	B	-	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	NL	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	NL	-	-	-	-	-
13	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
14	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
15	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
16	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
17	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	CAD	AG		AG	AG	AG		AG
18	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS

[Imagen 2] Matriz de acciones semánticas sin contemplar caminos de error

IB = Inicializar Buffer
C = Caracter
NL = New Line
“-” = null
KW = Verificar KeyWord
AG = Agregar Caracter
ID = Verificar Largo ID
F = Verificar Rango Float
CO = Verificar Comparador
CAD = Cadena

Dentro de esta matriz de transición de estados, había celdas que estaban incompletas porque en ciertos estados con determinados caracteres no había una transición válida.
Los casos no resueltos fueron tratados con acciones semánticas. Nos encontramos con 5 tipos de errores:

- Un salto de línea en una cadena, lo que resultaría en una cadena multilínea
- Un carácter aislado
- Integer mal escrito
- Sufijo del integer incorrecto
- Float mal escrito

Para estos casos se tomaron diferentes medidas, las cuales serán explicadas en el orden que fueron dados los errores.

- Se indicó un error al haber una cadena multilínea y asumimos que el programador se olvidó de cerrar la cadena, por lo que dejamos de leer y enviamos el token
- Se informó el error e ignoramos el carácter
- Se informó el error, se envió el integer que había sido detectado hasta el momento como token y se volvió a leer el carácter que nos produjo el error
- Se informó el error y se envió el integer que había sido detectado como token
- Se informó el error y se envió el flotante que había sido detectado, pero sin la parte del exponente.

Las acciones semánticas que trataban estos errores fueron agregados a la matriz anteriormente definida:

	l	L	d	_	%	+	-	"/"	*	{,},(,)	.	<	=	>	!	"	Bl/tab	nl	i	f	otro	$	:
0	IB	IB	IB	EC	EC	C	C	-	C	C	IB	IB	IB	IB	IB	IB	-	NL	IB	IB	C	C	IB
1	KW	AG	KW	AG	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW	KW
2	AG	ID	AG	AG	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	ID	AG	AG	ID	ID	ID
3	EI	EI	AG	EI	EI	EI	EI	EI	EI	EI	AG	EI	EI	EI	EI	EI	EI	EI	EI	EI	EI	EI	EI
4	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	EIS	INT	EIS	EIS	EIS	EIS
5	EC	EC	AG	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC
6	F	F	AG	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	AG	F	F	F
7	EF	EF	EF	EF	EF	AG	AG	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF
8	EF	EF	AG	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF
9	F	F	AG	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
10	B	B	B	B	-	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	NL	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	NL	-	-	-	-	-
13	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
14	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
15	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
16	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO	CO
17	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	AG	CAD	AG	ECM	AG	AG	AG	ECM	AG
18	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS

[Imagen 3] Matriz de acciones semánticas completa

EC: Error caracter
EI: Error Integer
EIS: Error Sufijo de Integer
ECM: Error Cadena Multilínea
EF: Error Float

Además fueron detectadas otros errores léxicos en algunas acciones semánticas:

- El valor de un integer en “Verificar Rango Integer”, que no puede ser mayor a 32768.
- Una asignación mal definida, si al inicio del token se detecta el caracter “:” y luego no se detecta el “=”.
- Una palabra reservada mal escrita.
- Constante flotante fuera de rango
- Longitud del identificador mayor a 20 (esto es un warning)

Analizador Sintactico

El analizador sintáctico verifica el cumplimiento de las reglas sintácticas. En él se define la gramática del lenguaje, la cual se rige por una serie de reglas sobre los tokens que son consumidos desde el analizador léxico.
Dicha demanda de tokens sirve para verificar si una secuencia dada pertenece al lenguaje o no. Es importante destacar que, además de las estructuras válidas del lenguaje, se deben definir las inválidas para informar errores y/o advertencias que ayudan al programador y para que se pueda seguir la compilación luego de consumir tokens erróneos. El analizador sintáctico consumirá tokens hasta que el código fuente se haya consumido por completo, aún en casos de errores sintácticos.
Lo primero que se hizo fue implementar la gramática mediante la sintaxis especificada para YACC. Esta gramática será compilada por un script externo que generará el código de la gramática de nuestro parser automáticamente.
Los no terminales utilizados en la gramática son:

- program: no terminal que da inicio a todo programa
- parámetros: permite construir una lista de identificadores para la declaración de variables.
- parametrosinvo: no terminal utilizado para definir parámetros reales en una llamada a procedimiento.
- tipo: utilizado para definir los diferentes tipos que puede ser una variable.
- asignación: no terminal que permite construir la estructura de una sentencia de asignación.
- expresión: es utilizado para dar inicio a operaciones aritméticas.
- término: no terminal utilizado para construir el resto de operación aritméticas y mantener procedencia de operadores.
- declarativas: no terminal del cual parten las sentencias procedure y declarativa
- cuerpowhile: cuerpo interno de la estructura procedure, a diferencia de cuerpo no incluye sentencias declarativas.
- factor: utilizado para definir las constantes, ya sean positivas o negativas
- boolean: utilizado para definir dos valores de verdad posibles, true y false
- procedure: no terminal que permite construir la estructura de un procedimiento
- cuerpo: no terminal principal del cual se crean las las sentencias declarativas y ejecutables
- ejecutable: utilizado para construir las sentencias de seleccion, salida, invocacion y asignacion
- declaración: no terminal utilizado para la definición de una o varias declaraciones de variables.
- control: utilizado para la construcción del bucle While.
- invocación: usado para definir los llamados a procedimientos
- salida: permite la definición de la sentencia de salida OUT.
- selección: no terminal que estructura la sentencia de selección IF.
- condición: permite la construcción de condiciones tanto para las sentencias IF como las While
- comparador: no terminal que define los tipos de símbolos de comparación permitidos por la gramática.
- lp: no terminal utilizado para definir parámetros formales en un procedimiento.

Se crearon 4 archivos de texto que van a contener, los errores que se produzcan tanto en el análisis léxico como en el sintáctico, los tokens que el léxico le devuelva al sintáctico, el contenido de la tabla de símbolos y por último uno que guarde todas las sentencias detectadas por el analizador sintáctico. En todos estos casos, se provee información detallada, ya sea del tipo de error o del tipo de sentencia o token detectado. También en el caso de los errores, añadiremos una aproximación sobre el número de línea en el cual se encuentran.
Para llevar una cuenta de los números de línea, se tiene una variable estática que cuenta cada vez que en el analizador léxico detecte un salto de línea. Además, en la gramática para poder hacer una aproximación de los números de líneas en los que se inician y terminan estructuras, o en las líneas que se produce un error; cada vez que se recibe un token del analizador léxico utilizando la variable *yylval* de tipo *ParserVal,* se utilizó el campo *ival* para almacenar el numero de linea en el que fue detectado ese token.

Errores sintácticos considerados por el compilador.

En la construcción del compilador, se vio que algunos errores en la parte sintáctica podían ser abordados de forma que el compilador pueda seguir analizando el resto del código y detectar otros errores, o procesar una mayor porción del código. Algunos de estos errores considerados son:

- Parámetros faltantes: cuando en una declaración de parámetros, ya sea como definición de parámetros, parámetros de invocación o en la definición de parámetros de un proceso, se encuentra que se tiene una coma sin que sea seguida de otro parámetro, se informa de este problema pero se lo permite como si no tuviera error.
- Estructuras ejecutables: utilizamos el no terminal "*error*" ya incluido por el *yacc*, para que si se detecta un error dentro de una de estas, se informe de este, y se busque un punto y coma (";"), como token de resincronización, ignorando la porción de código entre la definición de la sentencia ejecutable, y el token de resincronización.

Conclusión

Mediante la realización de este trabajo se pudo profundizar en el funcionamiento del análisis léxico y sintáctico de un compilador en la forma práctica. Se pudo ver todo lo que ofrece la herramienta yacc, para facilitar el desarrollo de un compilador. También, se tuvo que tomar decisiones a la hora de desarrollar el lenguaje descrito por la cátedra, mediante la definición de la matriz de transición de estados y la matriz de acciones semánticas, como también la definición de gramática del lenguaje y algunas funciones para tratar errores, que permitan poder analizar una mayor parte del código en presencia de algunos errores.